Computable Variants of AIXI
which are More Powerful than AIXItl

Susumu Katayama   SKATA@CS.MIYAZAKI-U.AC.JP
University of Miyazaki
1-1 Gakuenkibanadainishi
Miyazaki, Miyazaki 889-2192, Japan

Editor: Laurent Orseau

Abstract

This paper presents Unlimited Computable AI, or UCAI, that is a family of computable variants of AIXI. UCAI is more powerful than AIXItl, which is a conventional family of computable variants of AIXI, in the following ways: 1) UCAI supports models of terminating computation, including typed lambda calculi, while AIXItl only supports Turing machine with timeout $\tilde{t}$, which can be simulated by typed lambda calculi for any $\tilde{t}$; 2) unlike UCAI, AIXItl limits the program length to some $\tilde{l}$.

Keywords: AIXI, artificial general intelligence, exact real arithmetic, computability

1. Introduction

AIXI (Hutter, 2007) is an AI model for theoretical discussion of the limitations of AI. AIXI is expected to be universal. AIXI models the environment as a Turing machine and formalizes the interaction between the environment and the AI agent as a discrete-time *reinforcement learning* (e.g., Sutton and Barto, 1998) problem, that is an optimization problem under unknown environment. AIXI is not computable without approximation.

AIXItl and AIXI$\widetilde{tl}$ (Hutter, 2007) are computable variants of AIXI, but they have limitations on the description length of the environment and the computation time for each time step when computed by a sequential Turing machine. Especially, the limitation on the program description length may be a problem when dealing with the universe whose size is unknown beforehand.

In this paper, we introduce AI models that generalize the definition of AIXI to support a broader class of models of computation as the environment model than Turing machines. Then, we show the computability of our AI models adopting models of terminating computation such as typed λ calculi. Because the model of computation used by AIXItl and AIXI$\widetilde{tl}$ is "Turing machines with timeout $\tilde{t}$ within length $\tilde{l}$", which is a more limited class than structural recursive functions, our AI models cover a more general environment class. We claim that our introduction of such computable AI models which are more powerful than known models is useful for research on the theoretical limitations of computable AI models.

We prove two main theorems about Unlimited AI (UAI), that is our formalization of generalized AIXI, when using a model of terminating computation as the environment model.

The first theorem shows that the *action value function* (representing how valuable each action in each situation is) of UAI is computable to arbitrary precision. Roughly speaking, this result means that UAI is at least as computable as Q-learning (Watkins and Dayan, 1992): UAI and Q-learning

1

both select an action at each time step by $\arg\max$ operation on real-valued action value functions, but strictly speaking, the $\arg\max$ operation is not exactly computable if different actions may have the same action value, because a comparison of exactly the same real values requires comparisons of an infinite number of digits. Practically, however, Q-learning is considered computable, by giving up computation at some precision.

The other theorem shows that UAI becomes exactly computable including the $\arg\max$ operation with probability 1 by randomly selecting the least significant bits of each prior probability. We call the resulting AI model Unlimited Computable AI (UCAI).

Making AIXI computable involves solving the following three problems:

Problem 1 programs that are candidate models of the environment can enter infinite loops;

Problem 2 the agent needs to compute an infinite series because there are an infinite number of candidate models of the environment;

Problem 3 all operations are on real values.

AIXItl deals with the above problems in the following way:

1. it deals with Problem 1 by limiting the computation time by introducing a timeout;

2. it deals with Problem 2 by limiting the number of programs finite by limiting the length of programs.

Although Problem 3 must also be considered for strictly theoretical discussion, it is just ignored in the papers on AIXItl (e.g., Hutter, 2007), and it is not explicitly solved. However, if the set of rewards is limited to rational numbers, all the computation can be executed as operations on rational numbers, because the number of programs is finite in AIXItl. Throughout this paper, we assume that the reward can only take rational numbers.

On the other hand, our UCAI algorithm deals with the three problems in the following way:

1. by not limiting the environment model to Turing machines and permitting models of terminating computation, it deals with Problem 1 while supporting models of computation which are more powerful than Turing machines with any timeout;

2. it deals with Problem 2 and 3 without limiting the program length by applying the technology called *exact real arithmetic* (e.g., Boehm and Cartwright, 1990), which enables exact computation with real values under some limitations. Intuitively, the idea is to represent bounded real values as lazy infinite streams of digits.

The rest of this paper is organized in the following way. Section 2.2 introduces AIXI. Section 2.3 introduces exact real arithmetic and its implementation using lazy evaluation. Section 3 defines UCAI and proves that UCAI is exactly computable. Section 4 summarizes the results and discusses what the results mean.

2. Preparation

This section provides the definitions of the concepts used in this paper.

2.1 Finite List

Before defining AIXI, we define (finite) lists, because AIXI deals with the discrete time sequence. Infinite lists or streams will be defined in Section 2.3.

Definition 1 (Finite List) *A finite list is either the empty list $[]$ or the result of prepending an element to a finite list. $:$ denotes the prepending operator, corresponding to the cons function in Lisp.*

$:$ is right associative. For example, the list $x_1 : (x_2 : (x_3 : []))$ can be written as $x_1 : x_2 : x_3 : []$ by omitting the parentheses, which is the list consisting of x_1, x_2, and x_3 in this order.

$A*$ denotes the type of lists of A's for any type A in general.

For lists we define the concatenation operator $+\!\!\!+$, such that $(a_1 : a_2 : []) +\!\!\!+ (b_1 : b_2 : b_3 : []) = a_1 : a_2 : b_1 : b_2 : b_3 : []$.

Definition 2 (Concatenation)

$$[] +\!\!\!+ y = y$$
$$(x_1 : x) +\!\!\!+ y = x_1 : (x +\!\!\!+ y)$$

$+\!\!\!+$ is also right-associative.

2.2 AIXI

AIXI (Hutter, 2007) is an AI model for theoretical discussion of the limitations of AI. AIXI is expected to be universal.

Mostly, AIXI is based on the general reinforcement learning framework:

- at each time step, the learning agent and the environment interact via *perceptions* and *actions*: the agent chooses an action based on the interaction history and sends it to the environment, then, the environment chooses a perception based on the history and sends it to the agent, then again the agent chooses an action based on the new history and sends it to the environment, and so on;

- the perception includes the information of *reward*, which reflects how well the agent has been behaving; the agent's purpose is to maximize the *expected return*, that is the expectation of the total sum of future reward;

- the environment is unknown to the agent. It may even change in time.

AIXI models the environment as a Turing machine. It estimates the environment in the way that higher prior probabilities are assigned to simpler programs. At each time step, it selects the action which maximizes the expected return weighted by the belief assigned to each Turing machine.

Remark 3 *Turing machines and other models of computation also have the idea of the number of computation steps for measuring the computation time. Although we do not mention each computation step in this paper, we often use the term "interaction step" instead of "time step" in order to emphasize that we are not talking about computation steps.*

AIXI has the following parameters:

- the finite non-empty set of actions $\mathcal{A}$,

- the finite non-empty set of observations $\mathcal{O}$,

- the finite and bounded set of rewards $\mathcal{R} \subset [r_{\min}, r_{\max}]$, and

- the horizon function $m \in \mathbb{N} \to \mathbb{N}$ such that $m(k) \geq k$.

Although Hutter (2007) assumes non-negative rewards, we omit this limitation because this is requested only by AIXItl which is an AIXI approximation, and also because this is easily amendable. Also, we do not often mention $\mathcal{O}$, but instead, the set of perceptions $\mathcal{E} = \mathcal{O} \times \mathcal{R}$ and the projection function $r \in \mathcal{E} \to \mathcal{R}$. [1]

At each time step k, AIXI determines the action a_k from the interaction history based on the following equations. Firstly, the action value function $Q \in (\mathcal{A} \times \mathcal{E})* \times \mathcal{A} \to \mathbb{R}$ computing the expected return for each action based on the current history is defined as follows:

$$
\begin{aligned}
&Q(\text{æ}_{1..k-1}, a_k) \\
&= \sum_{e_k \in \mathcal{E}} \max_{a_{k+1} \in \mathcal{A}} \sum_{e_{k+1} \in \mathcal{E}} \max_{a_{k+2} \in \mathcal{A}} \dots \sum_{e_{m(k)-1} \in \mathcal{E}} \max_{a_{m(k)} \in \mathcal{A}} \sum_{e_{m(k)} \in \mathcal{E}} \\
&\left(\left(\sum_{t=k}^{m(k)} r(e_t) \right) \sum_{\substack{q \in T, \\ [\![q]\!](a_{1..m(k)})=e_{1..m(k)}}} \xi(q) \right)
\end{aligned}
$$

where e_k denotes the observation at time k. In general, $v_{m..n}$ denotes $v_m : v_{m+1} : \dots : v_{n-1} : v_n : [\,]$, i.e., the sequence from time m to time n of time-varying variable v. æ_k denotes (a_k, e_k) that is the action-perception pair at time k. Thus, $\text{æ}_{1..k-1}$ is the interaction history $(a_1, e_1) : \dots : (a_{k-1}, e_{k-1}) : [\,]$ at time k. Also note that

$$
a_{1..m(k)} = a_{1..k-1} \mathbin{+\!\!+} a_k : a_{k+1..m(k)}
$$

$$
e_{1..m(k)} = e_{1..k-1} \mathbin{+\!\!+} e_{k..m(k)}
$$

(because $1 \leq k \leq m(k)$), and that $a_{1..k-1}$, a_k, $a_{k+1..m(k)}$, $e_{1..k-1}$, and $e_{k..m(k)}$ are bound in the different ways.

T is the set of monotone Turing machines. $\xi \in T \to [0, 1]$ is the *universal prior* defined as

$$
\xi(q) = 2^{-l(q)} \tag{1}
$$

where $l(q)$ is the length of q in the prefix code. The universal prior is designed to prefer simple programs by assigning lower probabilities to longer programs.

$$
[\![q]\!]\, (a_{1..m(k)}) = e_{1..m(k)} \tag{2}
$$

---

1. Throughout this paper, we use '$\in$' instead of ':' even for functions for readability, in order to avoid the name collision against the ':' operator defined in Definition 1. Its fixity is weaker than $\to$, and thus $r \in \mathcal{E} \to \mathcal{R}$ means $r \in (\mathcal{E} \to \mathcal{R})$.

denotes the condition that q behaves interactively, taking a_1 as the input, returning e_1 as the output, taking a_2 as the input, returning e_2 as the output, and so on, and taking $a_{m(k)}$ as the input, and returning $e_{m(k)}$ as the output.

Based on the action value function Q defined above, AIXI always chooses the best action, i.e., the action $\dot{a}_k$ at time k after the actual interaction history $\ddot{æ}_{k-1}$ such that

$$\dot{a}_k = \arg\max_{a \in \mathcal{A}} Q(\ddot{æ}_{1..k-1}, a) \tag{3}$$

2.3 Exact Real Arithmetic and Lazy Evaluation

Exact real arithmetic (e.g., Boehm and Cartwright, 1990) is a set of techniques for effectively implementing exact computations over the real numbers. Because the set of real numbers is a continuum and each real number contains the information of an infinite number of digits, the reader may doubt if it is even possible. Actually, that is not a problem because the set of real numbers that can be uniquely defined by a finite program is countable.

There are two main kinds of approaches to representing exact real values.(Plume, 1998) One represents them as a function taking the precision returning an approximation to the precision (Boehm and Cartwright, 1990), and the other represents the mantissa as a lazy infinite stream of digits or other integral values. The latter stream-based approach has various representations, which are reviewed by Plume (1998).

Lazy functional languages such as Haskell adopt the lazy evaluation model and can deal with infinite data structures such as infinite lists (a.k.a. streams) and infinite trees. The idea of lazy evaluation is to postpone computation until it is requested. Even if the remaining computation will generate infinite data, the data structure can hold a *thunk*, or a description of the remaining computation, until a more precise description is requested.

Now we give a definition of a stream.

Definition 4 (Stream) *A* stream *is an infinite list. An infinite list is the result of appending an element in front of an infinite list using the cons* : *operator. (We use the same letter* : *for both finite and infinite lists.)*

Again, : *is right associative. For example, the stream* $x_1 : (x_2 : (x_3 : ...$ *can be written as* $x_1 : x_2 : x_3 : ...$ *by omitting the parentheses.*

$A\infty$ *denotes the type of streams of A's for any type A in general.*

This is an intuitive definition. More interested readers should read a functional programming textbook (e.g., Bird and Wadler, 1988).

In this paper, stream variables are suffixed with ∞ for readability, such as x_∞.

We prove the computability of our AIXI variants when using redundant binary (a.k.a. signed binary) stream representation. For simplicity, we normalize the set of rewards in order for all the computations to be executed within $(0, 1)$, and stick to fixed point computations.

Definition 5 (Fixed point redundant binary stream representation, FPRBSR) *Let* $x \in [-1, 1]$. *A fixed point redundant binary stream representation, or* FPRBSR *in short, of* x *is a stream* $x_1 : x_2 : x_3 : ...$ *where*

$$x = \sum_{i=1}^{\infty} 2^{-i} x_i$$

5

and
$$x_i \in \{-1, 0, 1\}$$

hold for each i.

$\llbracket \cdot \rrbracket \in \{-1, 0, 1\}\infty \to [-1, 1]$ *denotes the interpretation of the given FPRBSR, or conversion to* $[-1, 1]$.

$$\llbracket x_1 : x_2 : x_3 : ... \rrbracket = \sum_{i=1}^{\infty} 2^{-i} x_i$$

holds.

In order to make sure that such infinite computations generate results infinitely, namely, to make sure the computability to arbitrary precisions, it is enough to show that some initial element(s) (or digit(s), in the case of a stream of digits) and the data representing the remaining computation can be computed at each recursion.

Definition 6 (Digit-wise computability) *Function* f *returning an FPRBSR is* digit-wise computable *iff it infinitely generates the resulting stream, i.e., for any precision* p *there always exists the time* t *when the result is computed to the precision* p.

Digit-wise computability is the computability of FPRBSR. It means the value can be computed to any given precision. It is just called computability in the literature (e.g., Plume, 1998).

Digit-wise computability should not be confused with computability in the limit. Obviously, the former is stronger than the latter. The results which are obtained during the process of digit-wise computations are exact and definite, and can never be overwritten. On the other hand, computability in the limit does not care about the process nor guarantee any truth of the results obtained during the computation.

In order to prove that the function f is digit-wise computable, it is enough to show that f can be represented as

$$f(x) = g(x) : (h(f))(x)$$

We need to request that once the first elements of streams, or the most significant digits in the case of streams of digits, are computed and fixed, they must not change later by carrying. In fact, the usual binary representation does not always satisfy this rule: e.g., when we are adding $0.01010...$ to $0.00101...$, whether the most significant digit of the result is 0 or 1 may be undecided forever. Instead, we use the redundant binary representation which only requests finite times of carrying out for most arithmetic operations, for avoiding this problem.

The most serious limitation of exact real arithmetic is that comparisons between two exactly the same numbers do not terminate (unless we know that they are the same beforehand), because such comparisons mean comparisons between exactly the same infinite lists. We can still compare two different numbers to tell which is the greater.

In the defining equations of AIXI, this limitation only affects their $\arg\max$ operations. (It does not affect the $\max$ operations, as shown in Lemma 17.) This means that it is very difficult to exactly choose the best action when the best and the second best action values are almost the same. In practice, however, there may be cases where we can ignore such small differences and use fixed-precision floating point approximations instead of exact real numbers.

This paper proves that the action values of our AIXI generalization can be computed to arbitrary precisions for arbitrary prior distributions when a model of terminating computation instead of

a Turing machine is used (Theorem 7), and that the whole computation, including the $\arg\max$ operation, is exactly computable with probability 1 if we modify the least significant digits of probabilities of the prior distribution to random irrational values (Theorem 10).

3. Contributions

In this section, we introduce our AIXI variant which supports more powerful models of computation than what AIXItl uses and is still computable. For this, we start with generalizing AIXI to the necessary level, and then we specialize it to obtain computable models.

3.1 Generalizing AIXI

In this section, we generalize AIXI by generalizing the model of computation from monotone Turing machine to other models and generalizing the prior distribution function on it.

3.1.1 GENERALIZING THE MODEL OF COMPUTATION

AIXI uses universal prefix Turing machine, that executes any program in a prefix code. The program is usually written as an encoding of a Turing machine or in a Turing-complete language, but it can be written in a terminating language such as typed λ calculi and functional languages extending them.

Although the reader may think that using a Turing-incomplete model of computation is unacceptable, it should be better than AIXItl, because "Turing machines with a timeout", which AIXItl uses, can be simulated by typed λ calculi.

When using a terminating functional language, the effect of using *monotone* Turing machines can be achieved by using lazy I/O, i.e., by modeling the input and the output as streams. (Moreover, even when streams are not available, the same computational ability is achieved by supplying the interaction history instead of supplying only the current perception as the current input, though the efficiency is sacrificed.)

The idea of termination and lazy I/O can coexist. For example, Agda is a computer language equipped with both. The careful reader should recall that similarly monotone Turing machines may or may not enter an infinite loop while computing each output in reply to each input.

Extending the available set of models of computation to include λ calculi and functional languages has another bonus of enabling incremental learning by assigning biased prior probabilities to prioritize expressions with useful functionality.(Katayama, 2016) We do not discuss this further in this paper.

3.1.2 ASSIGNING PRIOR DISTRIBUTIONS

For any model of computation $\mathcal{M}$ not limited to Turing machines, its universal prior can be defined in the same way as Eq. (1):

$$\xi_1 \in \mathcal{M} \rightarrow [0, 1]$$
$$\xi_1(q) = 2^{-l(q)} \tag{4}$$

where $l(q)$ denotes the length of q in the prefix code. The prefix code should be a compressed one based on the grammar when considering efficiency, though usual programs in plain text are already

in prefix codes. Adequate selection of a concise prefix code helps to avoid grammatically-incorrect programs.

Note that there can be grammatically-correct, but type-incorrect programs, in the same way as non-terminating programs of Turing machines. Such programs should be selected with probability 0, and the actual probabilities for valid programs should be those divided by $\sum_{q \in \mathcal{V}} \xi_1(q)$, where $\mathcal{V}$ denotes the set of valid programs. Operationally, this is equivalent to retrying generation of syntax tree when Eq. (4) results in an invalid program. There is no problem, because the probability of generating a valid terminating program based on Eq. (4) is constant if the language is fixed, and because the result of Eq. (3) would not be affected by multiplication by a constant.

For any finite program $q \in \mathcal{M}$, $\xi_1(q)$ is rational, though the actual prior

$$\frac{\xi_1(q)}{\sum_{q \in \mathcal{V}} \xi_1(q)}$$

may not.

3.1.3 FORMALIZING GENERALIZED AIXI

UAI, or *Unlimited AI*, is our formalization of generalized AIXI supporting the above generalizations. A UAI has the following parameters:

- the non-empty finite set of actions $\mathcal{A}$,

- the non-empty finite set of observations $\mathcal{O}$,

- the finite and bounded set of rational rewards $\mathcal{R} \subset [r_{\min}, r_{\max}] \wedge \mathbb{Q}$ where $r_{\min} < r_{\max}$ and $r_{\min}, r_{\max} \in \mathbb{Q}$,

- the horizon function $m \in \mathbb{N} \to \mathbb{N}$ such that $m(k) \geq k$,

- the model of computation $\mathcal{M}$, which can be viewed as a set of programs taking a lazy list of actions and returning a lazy list of perceptions,

- the prior distribution function $\rho \in \mathcal{M} \to [0, 1]$

UAI$(\mathcal{A}, \mathcal{O}, \mathcal{R}, m, \mathcal{M}, \rho)$ denotes UAI with the above parameters. The definition of the set of perceptions $\mathcal{E} = \mathcal{O} \times \mathcal{R}$ and the projection function $r \in \mathcal{E} \to \mathcal{R}$ are the same as those of AIXI's.

The action value function $Q \in (\mathcal{A} \times \mathcal{E})* \times \mathcal{A} \to \mathbb{R}$ of UAI$(\mathcal{A}, \mathcal{O}, \mathcal{R}, m, \mathcal{M}, \rho)$ is defined as follows:

$$
\begin{aligned}
&Q(\text{æ}_{1..k-1}, a_k) \\
&= \sum_{e_k \in \mathcal{E}} \max_{a_{k+1} \in \mathcal{A}} \sum_{e_{k+1} \in \mathcal{E}} \max_{a_{k+2} \in \mathcal{A}} ... \max_{a_{m(k)} \in \mathcal{A}} \sum_{e_{m(k)} \in \mathcal{E}} \\
&\left(\left(\sum_{t=k}^{m(k)} r(e_t) \right) \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \rho(q) \right)
\end{aligned}
\tag{5}
$$

where

$$[\![q]\!]\,(a_{1..m(k)}) = e_{1..m(k)}$$

denotes the condition that q behaves interactively in the same way as in the case of Eq. (2), if q is an interactive program such as that of monotone Turing machines and an implementation using lazy I/O. $\mathcal{M}$ can be a model of computation without interaction; in such cases, q' such that

$$q'(a_{1..1}, [\,]) = e_1$$
$$q'(a_{1..2}, e_{1..1}) = e_2$$
$$\vdots \qquad\qquad \vdots$$
$$q'(a_{1..m(k)}, e_{1..m(k)-1}) = e_{m(k)}$$

has to be implemented, which requires recomputation of the states of the environment. [2]

Based on the action value function Q defined above, $\text{UAI}(\mathcal{A}, \mathcal{O}, \mathcal{R}, m, \mathcal{M}, \rho)$ always chooses the best action, i.e., the action $\dot{a}_k$ at time k after the actual interaction history $\ddot{æ}_{k-1}$ such that

$$\dot{a}_k = \arg\max_{a \in \mathcal{A}} Q(\ddot{æ}_{1..k-1}, a) \tag{6}$$

AIXI can be represented as $\text{UAI}(\mathcal{A}, \mathcal{O}, \mathcal{R}, m, T, \xi)$ using the set of monotone Turing machines T and the universal prior ξ. Although AIXI does not explicitly limit the set of rewards to rational numbers unlike UAI, how to represent real numbers is not discussed by papers on AIXI.

3.1.4 COMPUTABILITY OF UAI

Is UAI implementable using exact real arithmetic? Our conclusion is that action values are digit-wise computable, and the only part that can cause an infinite loop is the $\arg\max$ operation, when using a model of terminating computation as the environment model. One may think that the infinite summation may also cause an infinite loop, but this is not the case actually if the set of rewards is bounded and does not change with time, because the summation is bounded by a geometric series, and thus more and more digits become fixed from the most significant ones as the computation proceeds.

The following Theorem 7 clarifies the above claim.

Theorem 7 (Computability of the action value function of UAI) *The action value function* $Q(æ_{1..k-1}, a)$ *of* $\text{UAI}(\mathcal{A}, \mathcal{O}, \mathcal{R}, m, \mathcal{M}, \rho)$ *defined by Eq. (5) is digit-wise computable if* $\mathcal{R} \subset \left[0, \frac{1}{m(k)-k+1}\right)$ *holds and* $\mathcal{M}$ *is a model of terminating computation.*

See Appendix A.3 for the proof of Theorem 7.

This theorem shows that the action-value function of UAI is exactly computable, or computable to arbitrary precision, provided that the set of possible environments is a set of total functions. We could not prove that the $\arg\max$ operation over the exact action values is computable for general priors such as the AIXI's ξ, because that operation involves comparisons between real numbers which can be exactly the same. The reader should notice that action selection of Q-learning also

---

2. We wrote $a_{1..1}$ and $e_{1..1}$ instead of a_1 and e_1 respectively, because they are actually *lists of* actions and perceptions, i.e., $a_1 : [\,]$ and $e_1 : [\,]$.

involves such $\arg\max$ operation over the real numbers. In other words, this theorem proves that UAI is as computable as Q-learning.

Theorem 7 is proved for the case of $\mathcal{R} \subset \left[0, \frac{1}{m(k)-k+1}\right)$; thanks to the following Lemma 8, this does not limit the applicability of the theorem.

Lemma 8 (Linearity) *For a positive real number p and a real number s, let Q' be the Q value obtained by replacing r in Eq. (5) with r' where $r'(e) = pr(e) + s$, i.e.,*

$$Q'(\mathit{æ}_{1..k-1}, a_k)$$
$$= \sum_{e_k \in \mathcal{E}} \max_{a_{k+1} \in \mathcal{A}} \sum_{e_{k+1} \in \mathcal{E}} \max_{a_{k+2} \in \mathcal{A}} \ldots \max_{a_{m(k)} \in \mathcal{A}} \sum_{e_{m(k)} \in \mathcal{E}}$$
$$\left(\left(\sum_{t=k}^{m(k)} r'(e_t) \right) \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \rho(q) \right) \quad (7)$$

Then, for a real number $c(\mathit{æ}_{1..k-1})$,

$$Q'(\mathit{æ}_{1..k-1}, a) = pQ(\mathit{æ}_{1..k-1}, a) + sc(\mathit{æ}_{1..k-1}) \quad (8)$$

holds.

See Appendix A.3 for the proof of Lemma 8.

Corollary 9 *For any positive real number p and any real number s, the value of Eq. (6) does not change when r in Eq. (5) is replaced with r' where $r'(e) = pr(e) + s$. In other words, for Q' defined by Eq. (7)*

$$\arg\max_{a \in \mathcal{A}} Q(\mathit{æ}_{1..k-1}, a) = \arg\max_{a \in \mathcal{A}} Q'(\mathit{æ}_{1..k-1}, a)$$

Proof Self-explanatory from Lemma 8. ∎

Thanks to lazy evaluation, UAI automatically omits unnecessary executions of environment candidate programs which do not affect the result of the $\arg\max$ operation. How much computation time is saved is unpredictable, depending on the history. However, UAI can return the result in the precision it has at the deadline, if there exists the deadline for each interaction step. It can have the $\arg\max$ set based on the first digit of Q values, the $\arg\max$ set based on the first two digits of Q values, and so on, and randomly select from the most precise $\arg\max$ set at the deadline.

3.2 Making UAI Fully Computable

Now we show that even the $\arg\max$ part of UAI can be made implementable by modifying the prior distribution adequately. We call the resulting AI model *Unlimited Computable AI (*UCAI).

Because AIXI assigns less plausibility to longer programs, it is possible (and in fact, it is really the case) that only a finite number of the most plausible programs affect the result of the decision making by the $\arg\max$ operator over the action-values, and the remaining infinitely many programs

do not affect it. The action-values need to be computed only to the precision where we can tell the difference between them. All we have to take care of is not to compare exactly the same values, because the comparison between the same values to the precision where they make difference results in an infinite loop.

Thus, we can concentrate on avoiding comparisons between action values which may be the same. We can do two things:

- compare two values that are known to be different beforehand, and tell which is the greater;

- skip comparison of two values that are known to be the same beforehand, and say that they are the same.

In other words, it is enough to know whether two action values are the same or not before comparison.

Accidental coincidences between action values may be avoided by

- slightly modifying the lower bits of prior distribution to consist of irrational numbers,

- limiting $\mathcal{R}$ to positive numbers in order to avoid the sum of rewards happening to be 0.

3.2.1 PRIORS FOR MAKING DIFFERENCE IN VALUES

This section discusses how to assign prior probabilities in order to make the whole things computable.

Our idea is to subtract small randomized number $\delta d(\eta(q))$ from the normal rational prior for each q, where $\eta(q)$ denotes the natural number representation of q. The resulting prior $\xi_2(q)$ can be represented as

$$\xi_2(q) = \xi_1(q)(1 - \delta d(\eta(q))) \tag{9}$$

by using ξ_1 of Eq. (4).

η can be defined as adding 1 to the left of reversed q and interpreting the result as a binary number, i.e.,

$$\eta([]) = 1$$
$$\eta(b : x) = b + 2\eta(x)$$

if q is defined as a list of bits, or $\{0, 1\}*$. Then,

$$l(q) = \lfloor \log_2 \eta(q) \rfloor$$

and thus

$$\xi_1(q) = 2^{-\lfloor \log_2 \eta(q) \rfloor}$$
$$\xi_2(q) = 2^{-\lfloor \log_2 \eta(q) \rfloor}(1 - \delta d(\eta(q)))$$

δ is a positive small rational number less than 1; a reasonable choice is 2^{-64}, which makes the additional term insignificant for those who do not care about the difference between fixed precision floating point approximations and real numbers.

The function d assigns a different infinite binary fraction randomly for each natural number i. $d(i)$ can be obtained by splitting an ideally random bit stream source for i times:

$$d(i) = [\![d'(i)]\!]$$
$$d'(i) = \tau_0(split(\underbrace{\tau_1(split(...\tau_1(split(\langle\text{random bit stream}\rangle)))...)))}_{i}$$

where τ_0 and τ_1 are projection functions

$$\tau_0(x, y) = x, \qquad\qquad\qquad \tau_1(x, y) = y$$

and $split \in \{0, 1\}\infty \to (\{0, 1\}\infty \times \{0, 1\}\infty)$ is a function that splits a random stream into two random streams. Ideal acyclic random streams can be split by the leap-frog method, though practically, pseudo-random number generators must be split carefully.(e.g., Claessen and Pałka, 2013) In this paper, we show the computability of UCAI with probability 1, assuming that an ideal random bit stream source is available.

Now we can prove the following theorem:

Theorem 10 (Computability of UCAI) *Let $\mathcal{M}$ be a model of computation that only includes terminating programs and has a conditional construct. Then, $\text{UAI}(\mathcal{A}, \mathcal{O}, \mathcal{R}, m, \mathcal{M}, \xi_2)$ is computable with probability 1 for ξ_2 in Eq. (9).*

See Appendix A.3 for the proof of Theorem 10.

4. Conclusions

This paper proposed AIXI variants supporting a broader class of the environment than AIXI*tl*, and proved theorems on their computability.

When considering the real-world interaction, the processing time for each interaction step should be considered limited, even if we permit the discrete-time model. In this sense, a timeout is a natural idea, and it is understandable to limit the program length, considering that the information accessible within a limited time is limited. However, considering that the real-world which has vast space is highly parallel, AIXI*tl* which models the environment using a sequential model of computation is not necessarily the best selection.

On the other hand, UCAI needs to simulate the environment, many times at each interaction step. It is unnatural to think that a UCAI agent as a computer is as parallel as the environment.

Still, we think UCAI is more powerful than AIXI*tl* even when there is a deadline at each interaction step. In the case of AIXI*tl*, each environment candidate program timeouts at time $\tilde{t}$, and there are $2^{\tilde{l}}$ candidate programs. In the case of UCAI, on the other hand, there is no timeout for each environment candidate, and a UCAI agent only needs to timeout at the actual deadline t', and thus more programs can be tried. The main advantage of UCAI comes from the fact that UCAI automatically omits execution of longer environment candidate programs when shorter ones cause a difference in the action values, thanks to lazy evaluation.

One question is whether we need to use a Turing complete model for modeling the behavior of the environment at each interaction step, i.e., whether there is a case where AIXI should be used over UCAI. We think UCAI is enough for the following reasons:

1. indeed, this world can simulate Turing machines with finite tape, but requesting them to terminate within one interaction step is not a reasonable idea; we still request that the environment can simulate Turing machines by holding physical tapes;

2. AIXI does not adequately model the real world because it permits incomputable agents.

For the purpose of satisfying Item 1 only, it is enough if at each interaction step the environment model can compute the finite-to-finite map implementing the set of quadruples of Turing machines, even with a tabular representation without loops. MC-AIXI(FAC-CTW) (Veness et al., 2011), which approximates AIXI more aggressively than AIXItl, does not model the environment as a program, but instead, it uses Context Tree Weighting (CTW) (Willems, Shtarkov, and Tjalkens, 1995), that is essentially the tabular representation using the PATRICIA tree.(Katayama, 2016) We should note that tables cannot generalize, though.

Acknowledgements

Appendix A. The Detailed Proofs

This section gives the proofs of the theorems. In Appendix A.1 we prove several utility lemmas on digit-wise computabilities of simple operators on FPRBSR's. In Appendix A.2 we prove other lemmas that are necessary for proving the main theorems. Then, in Appendix A.3 the main theorems are proved.

A.1 Digit-wise Computabilities of Operations on FPRBSR's

Because the sum of two FPRBSR's may overflow, firstly we show that their average is digit-wise computable.

Lemma 11 (Average is digit-wise computable) *If x_∞ and y_∞ are both digit-wise computable FPRBSR's, then their average $[\![x_\infty]\!] \oplus [\![y_\infty]\!] = \frac{[\![x_\infty]\!] + [\![y_\infty]\!]}{2}$ is also digit-wise computable as an FPRBSR.*

Proof The following algorithm $\oplus_\infty$ computes $[\![x_\infty]\!] \oplus [\![y_\infty]\!]$ as an FPRBSR:

$$(x_1 : x_2 : x_{3\infty}) \oplus_\infty (y_1 : y_2 : y_{3\infty})$$
$$= \begin{cases} \frac{x_1+y_1}{2} : c : s, & \text{if } x_1 + y_1 \in \{-2, 0, 2\} \\ -1 : c+1 : s, & \text{if } x_1 + y_1 = -1 \text{ and } x_2 + y_2 < 0 \\ 0 : c-1 : s, & \text{if } x_1 + y_1 = -1 \text{ and } x_2 + y_2 \geq 0 \\ 0 : c+1 : s, & \text{if } x_1 + y_1 = 1 \text{ and } x_2 + y_2 < 0 \\ 1 : c-1 : s, & \text{if } x_1 + y_1 = 1 \text{ and } x_2 + y_2 \geq 0 \end{cases} \quad (10)$$

where $c : s = (x_2 : x_{3\infty}) \oplus_\infty (y_2 : y_{3\infty})$.

The idea of Eq. (10) is as follows:

- if $x_1 + x_2$ is even, then the average of x_1 and y_1 does not affect less significant digits, and thus they can be computed straightforwardly;

- if $x_1 + x_2 = -1$, then $1/4$ must be subtracted from the result; this can be achieved by either subtracting 1 from the second element of the result or subtracting 1 from the first element and adding 1 to the second element; the algorithm conditions on the sign of $x_2 + y_2$ in order to keep each digit within $\{-1, 0, 1\}$ without carrying out;

- if $x_1 + x_2 = 1$, then $1/4$ must be added to the result; this can be achieved by either adding 1 to the second element of the result, or adding 1 to the first element and subtracting 1 from the second element; again, the algorithm conditions on the sign of $x_2 + y_2$ in order to keep each digit within $\{-1, 0, 1\}$ without carrying out;

$\oplus_\infty$ defined above is digit-wise computable, because each recursive call of $\oplus_\infty$ determines one digit. $\blacksquare$

Lemma 12 (Sum is digit-wise computable) *If x_∞ and y_∞ are both digit-wise computable FPRBSR's and their average $[\![x_\infty]\!] \oplus [\![y_\infty]\!]$ is within $(-1/2, 1/2)$, then their sum $[\![x_\infty]\!] + [\![y_\infty]\!]$ is also digit-wise computable as an FPRBSR.*

Proof FPRBSR's within $(-1/2, 1/2)$ can be doubled by the following digit-wise computable function *double*.

$$
\begin{aligned}
double(0 : x_{2\infty}) &= x_{2\infty} \\
double(1 : x_{2\infty}) &= f(x_{2\infty}) \\
double(-1 : x_{2\infty}) &= g(x_{2\infty}) \\
f(0 : x_{2\infty}) &= 1 : f(x_{2\infty}) \\
f(-1 : x_{2\infty}) &= 1 : x_{2\infty} \\
g(0 : x_{2\infty}) &= -1 : g(x_{2\infty}) \\
g(1 : x_{2\infty}) &= -1 : x_{2\infty}
\end{aligned}
$$

The above functions implement the following real computations for x_∞, y_∞, and z_∞ such that $[\![x_\infty]\!] \in (-1/2, 1/2)$, $[\![y_\infty]\!] \in [-1, 0)$, and $[\![z_\infty]\!] \in (0, 1]$:

$$
\begin{aligned}
[\![double(x_\infty)]\!] &= 2\,[\![x_\infty]\!] \\
[\![f(y_\infty)]\!] &= 1 + [\![y_\infty]\!] \\
[\![g(z_\infty)]\!] &= -1 + [\![z_\infty]\!]
\end{aligned}
$$

Thus, $[\![x_\infty]\!] + [\![y_\infty]\!]$ can be computed as $x_\infty +_\infty y_\infty = double(x_\infty \oplus_\infty y_\infty)$. $\blacksquare$

Likewise, multiplication of redundant binary representations is known to be computable.(Plume, 1998)

Lemma 13 (Product is digit-wise computable) *If x_∞ and y_∞ are both digit-wise computable FPRBSR's, then their product $[\![x_\infty]\!]\,[\![y_\infty]\!]$ is also digit-wise computable as an FPRBSR.*

We also need to show that the maximum of two FPRBSR's is digit-wise computable.

For x and y in the non-redundant binary representation, $\max\{x, y\}$ can be computed by adopting digits of whichever of x and y until we know which is greater, and then use the digits from the greater of the two. This algorithm does not work correctly for x and y in the redundant binary representation, because we cannot tell which is greater by only comparing digits (e.g., $[\![1 : -1 : 1 : x_\infty]\!] = [\![1 : 0 : -1 : x_\infty]\!] = [\![0 : 1 : 1 : x_\infty]\!]$).

Our solution uses the fact that

$$\max\{x, y\} = \frac{x + y + |x - y|}{2} = \frac{x + y + |x + (-y)|}{2}$$

We need to show that negation and taking the absolute value are digit-wise computable.

Lemma 14 (Negation is digit-wise computable) *If x_∞ is a digit-wise computable FPRBSR, then its negation $-[\![x_\infty]\!]$ is also digit-wise computable as an FPRBSR.*

Proof The following algorithm $-_\infty(\cdot)$ computes $-[\![x_\infty]\!]$ as an FPRBSR:

$$-_\infty(y : y_\infty) \quad = \quad -y : -_\infty(y_\infty)$$

∎

Lemma 15 (The absolute value is digit-wise computable) *If x_∞ is a digit-wise computable FPRBSR, then its absolute value $|[\![x_\infty]\!]|$ is also digit-wise computable as an FPRBSR.*

Proof The following algorithm $|\cdot|_\infty$ computes $|[\![x_\infty]\!]|$ as an FPRBSR:

$$
\begin{aligned}
|0 : y_\infty|_\infty &= 0 : |y_\infty|_\infty \\
|1 : y_\infty|_\infty &= 1 : y_\infty \\
|-1 : y_\infty|_\infty &= 1 : -_\infty(y_\infty)
\end{aligned}
$$

∎

Lemma 16 (The binary $\max$ is digit-wise computable) *If x_∞ and y_∞ are both digit-wise computable FPRBSR's, then their maximal value $\max\{[\![x_\infty]\!], [\![y_\infty]\!]\}$ is also digit-wise computable as an FPRBSR.*

Proof Since
$$\frac{\max\{x, y\}}{2} = (x \oplus y) \oplus |x \oplus (-y)|$$

$\frac{\max\{x,y\}}{2}$ is digit-wise computable.

Since $\max\{x, y\}$ is either x or y, it can be represented in FPRBSR, and $\frac{\max\{x,y\}}{2} \in (-1/2, 1/2)$. Therefore, $\max\{[\![x_\infty]\!], [\![y_\infty]\!]\}$ can be computed as $(x_\infty \oplus_\infty y_\infty) +_\infty |x_\infty \oplus_\infty -_\infty(y_\infty)|_\infty$. ∎

Lemma 17 ($\max$ over sets is digit-wise computable.) *If X is a finite non-empty set of digit-wise computable FPRBSR's, then $\max X$ is also a digit-wise computable FPRBSR.*

Proof Let x_∞ and y_∞ be FPRBSR's.

If X is a finite and non-empty set of digit-wise computable FPRBSR's, $\max X$ can be computed by a finite number of binary $\max$ operations in the following way:

$$\max\{x_\infty\} = x_\infty$$
$$\max(\{x_\infty, y_\infty\} + Y) = \max(\{\max\{x_\infty, y_\infty\}\} + Y)$$

where $+$ over sets denotes the direct sum of two sets. ∎

Definition 18 (Comparison) Comparison *of two values is either $(<)$, $(\equiv)$, or $(>)$, where 'values' can be digits, FPRBSR's, or tuples of them. Comparison ? (except that of FPRBSR's) is defined in the following way, using the usual order relations $<$ and $>$ and the equality relation $=$:*

$$x \,?\, y = \begin{cases} (<), & \text{if } x < y \\ (\equiv), & \text{if } x = y \\ (>), & \text{if } x > y \end{cases}$$

Lemma 19 (Comparison of two different reals is computable) *If x_∞ and y_∞ are both digit-wise computable FPRBSR's and we know $[\![x_\infty]\!] \neq [\![y_\infty]\!]$ beforehand, then, their comparison $[\![x_\infty]\!] \,?\, [\![y_\infty]\!]$ is computable.*

Proof $[\![x_\infty]\!] \oplus (-[\![y_\infty]\!])$ is digit-wise computable. The comparison can be computed as $x_\infty \,?\, y_\infty = cmp(x_\infty \oplus_\infty -_\infty(y_\infty))$ using cmp defined as follows:

$$cmp(0 : z_\infty) = cmp(z_\infty)$$
$$cmp(-1 : z_\infty) = (<)$$
$$cmp(1 : z_\infty) = (>)$$

∎

Lemma 20 ($\arg\max$ of a monomorphism over a finite set is computable.) *If X is a finite non-empty set and $f \in X \to R$ is a monomorphism which is digit-wise computable using FPRBSR, then, $\arg\max_{x \in X} f(x)$ is computable.*

Proof Let $f_\infty \in X \to \{-1, 0, 1\}_\infty$ be a function that computes f, i.e., $\forall x \in X.\, [\![f_\infty(x)]\!] = f(x)$.

$\arg\max_{x \in X} f(x)$ can be computed by the following algorithm:

$$\arg\max_{x \in \{a\}} f(x) = a$$

$$\arg\max_{x \in \{a,b\}+Y} f(x) = \begin{cases} \arg\max_{x \in \{a\}+Y} f(x), & \text{if } f_\infty(a) \,?\, f_\infty(b) = (>); \\ \arg\max_{x \in \{b\}+Y} f(x), & \text{if } f_\infty(a) \,?\, f_\infty(b) = (<). \end{cases}$$

∎

A.2 digit-wise computability of Diminishing Series

In this section, we provide lemmas and their proofs about digit-wise computability of infinite series of positive values diminishing exponentially. Their main purpose is to be applied to the infinite summation in Eq. (5).

Lemma 21 (Diminishing series by 2^{-2} are digit-wise computable.) *If $0 \leq [\![x_\infty(i)]\!] < 1$ holds and $x_\infty(i)$ is a digit-wise computable FPRBSR for all positive integer i, then, $\sum_{i=1}^{\infty} 2^{-2i} [\![x_\infty(i)]\!]$ is digit-wise computable.*

Proof We show that $\sum_{i=1}^{\infty} 2^{-2i} [\![x_\infty(i)]\!]$ is digit-wise computable as $0 : S(1)$ using the function S defined as follows:

$$S(i) = \begin{cases} 1 : ((-1 : s_\infty) +_\infty 0 : S(i+1)), & \text{if } s = 1, \\ 0 : ((s : s_\infty) +_\infty 0 : S(i+1)), & \text{if } s \in \{-1, 0\} \end{cases} \tag{11}$$

$$\text{where } s : s_\infty = x_\infty(i).$$

Note that $s = -1$ suggests $s_\infty = 1 : s_\infty$ because $[\![x_\infty(i)]\!] \geq 0$. By definition, $[\![S(i)]\!] \geq 0$ holds. Obviously, S is digit-wise computable.

Now we show that

$$\sum_{i=1}^{\infty} 2^{-2i} [\![x_\infty(i)]\!] = [\![0 : S(1)]\!]$$

When $x_\infty(i) = 1 : s_\infty$, from the first case of Eq. (11), we obtain

$$\begin{aligned}[\![S(i)]\!] &= [\![1 : ((-1 : s_\infty) +_\infty 0 : S(i+1))]\!] \\ &= 2^{-1}(1 + [\![(-1 : s_\infty) +_\infty 0 : S(i+1)]\!]) \\ &= 2^{-1}(1 + [\![(-1 : s_\infty)]\!] + [\![0 : S(i+1)]\!]) \\ &= 2^{-1}([\![(1 : s_\infty)]\!] + [\![0 : S(i+1)]\!]) \\ &= 2^{-1} [\![x_\infty(i)]\!] + 2^{-2} [\![S(i+1)]\!] \end{aligned} \tag{12}$$

Likewise, for $x_\infty(i) = s : s_\infty$ where $s \in \{-1, 0\}$ from the second case of Eq. (11),

$$\begin{aligned}[\![S(i)]\!] &= [\![0 : ((s : s_\infty) +_\infty 0 : S(i+1))]\!] \\ &= 2^{-1} [\![x_\infty(i)]\!] + 2^{-2} [\![S(i+1)]\!] \end{aligned} \tag{13}$$

For both cases, from Eqs. (12) and (13),

$$[\![S(i)]\!] = 2^{-1} [\![x_\infty]\!] + 2^{-2} [\![S(i+1)]\!] \tag{14}$$

Thus, by applying Eq. (14) repeatedly,

$$\begin{aligned}[\![0 : S(1)]\!] &= 2^{-2} \left([\![x_\infty(1)]\!] + 2^{-2} \left([\![x_\infty(2)]\!] + 2^{-2} \left([\![x_\infty(3)]\!] + ... \right) \right) \right) \\ &= \sum_{i=1}^{\infty} 2^{-2i} [\![x_\infty(i)]\!] \end{aligned}$$

∎

Lemma 22 *Let $\rho : \mathcal{X}* \to [0, 1]$ a digit-wise computable probability function over finite lists of $\mathcal{X}$'s. Also, let P a computable predicate over such lists. Then, $\sum_{p:P(p)} \rho(p)$ is digit-wise computable.*

Proof Since ρ is a probability function,

$$\sum_p \rho(p) = 1$$

holds. Thus,

$$\sum_{p:P(p)} \rho(p) \leq 1$$

By reorganizing the summation from the shortest increasing the length, the left-hand side can be rewritten to

$$\sum_{k=1}^{\infty} \sum_{p:l(p)=k,P(p)} \rho(p) \leq 1$$

$$\sum_{k=1}^{\infty} \sum_{p:l(p)=k} \rho(p) = 1$$

where $l(p)$ denotes the length of p.

Now let

$$R(i) = \sum_{p:l(p)=i,P(p)} \rho(p)$$

$$R'(i) = \sum_{p:l(p)=i} \rho(p)$$

Then,

$$R(i) \leq R'(i)$$

$$\sum_{k=1}^{\infty} R(k) \leq 1$$

$$\sum_{k=1}^{\infty} R'(k) = 1 \tag{15}$$

and $R(i)$ and $R'(i)$ are digit-wise computable because they consist of finite summations and digit-wise computable computations. Let $R_\infty(i)$ and $R'_\infty(i)$ be FPRBSR representations of $R(i)$ and $R'(i)$ respectively.

Because the left-hand side of Eq. (15) is well-defined,

$$\lim_{i \to \infty} \sum_{k=i}^{\infty} R'(k) = 0$$

18

holds. In other words, for any $\epsilon > 0$ there exists a natural number $n > 0$ that satisfies $\forall j > n . \sum_{k=j}^{\infty} R'(k) < \epsilon$. Thus, for some Skolem function f that takes $\epsilon > 0$ and returns such a natural number $n > 0$,

$$\sum_{k=f(2^{-2i})}^{\infty} R'(k) < 2^{-2i}$$

holds.

There are infinite candidates for f because we need not choose the minimal n. Let

$$g(i) = f(2^{-2i})$$

Then, we can choose the following computable implementation of g:

$$\begin{aligned}
(g(0), s_\infty(0)) &= (1, 0_\infty) \\
(g(i), s_\infty(i+1)) &= g'(i, g(i), s_\infty(i)), i \geq 0 \\
g'(i, n, 0 : s'_\infty) &= g'(i, n+1, 0 : s'_\infty +_\infty R'_\infty(n)) \\
g'(i, n, 1 : s'_\infty) &= \begin{cases} (n+1, 1 : s'_\infty), & \text{if } s'_\infty \text{ is prefixed with } 2i \text{ 0's;} \\ g'(i, n+1, 1 : s'_\infty +_\infty R'_\infty(n)), & \text{otherwise.} \end{cases}
\end{aligned}$$

where

$$0_\infty = 0 : 0_\infty$$

If we define x as

$$x(k) = \sum_{i=g(k)}^{g(k+1)-1} R(i)$$

then, x is a sequence that diminishes by the rate of 2^{-2}. Moreover, each $x(k)$ is digit-wise computable because it can be computed from finite times of additions. Therefore, from Lemma 21,

$$\begin{aligned}
\sum_{i=1}^{\infty} R(i) \\
&= \sum_{i=1}^{g(1)-1} R(i) + \sum_{i=g(1)}^{g(2)-1} R(i) + \sum_{i=g(2)}^{g(3)-1} R(i) + ... + \sum_{i=g(k)}^{g(k+1)-1} R(i) + ... \\
&= \sum_{i=1}^{g(1)-1} R(i) + x(1) + x(2) + ...
\end{aligned}$$

is digit-wise computable because it is a series diminishing by 2^{-2}. ∎

A.3 Proofs of the Main Theorems

Proof [Proof of Lemma 8]

$$Q'(\text{æ}_{1..k-1}, a_k)$$
$$= \sum_{e_k \in \mathcal{E}} \max_{a_{k+1} \in \mathcal{A}} \sum_{e_{k+1} \in \mathcal{E}} \max_{a_{k+2} \in \mathcal{A}} \dots \max_{a_{m(k)} \in \mathcal{A}} \sum_{e_{m(k)} \in \mathcal{E}}$$
$$\left(\left(\left(p \left(\sum_{t=k}^{m(k)} r(e_t) \right) + s(m(k) - k + 1) \right) \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \rho(q) \right) \right.$$
$$= \sum_{e_k \in \mathcal{E}} \max_{a_{k+1} \in \mathcal{A}} \sum_{e_{k+1} \in \mathcal{E}} \max_{a_{k+2} \in \mathcal{A}} \dots \max_{a_{m(k)} \in \mathcal{A}} \sum_{e_{m(k)} \in \mathcal{E}}$$
$$\left(p \left(\sum_{t=k}^{m(k)} r(e_t) \right) \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \rho(q) \right.$$
$$\left. + s(m(k) - k + 1) \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \rho(q) \right)$$

Now, for each $a_{m(k)}$ there always exists only one $e_{m(k)}$, because $\mathcal{M}$ is a model of terminating computation. Thus,

$$\sum_{e_{m(k)} \in \mathcal{E}} \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \rho(q) = \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)-1}) = e_{1..m(k)-1}}} \rho(q)$$

More generally,

$$\sum_{e_{t+1} \in \mathcal{E}} \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..t+1}) = e_{1..t+1}}} \rho(q) = \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..t}) = e_{1..t}}} \rho(q)$$

20

Thus,

$$
\begin{aligned}
Q'&(\text{æ}_{1..k-1}, a_k)\\
&= p\left(\sum_{e_k\in\mathcal{E}}\max_{a_{k+1}\in\mathcal{A}}\sum_{e_{k+1}\in\mathcal{E}}\max_{a_{k+2}\in\mathcal{A}}\ldots\max_{a_{m(k)}\in\mathcal{A}}\sum_{e_{m(k)}\in\mathcal{E}}\right.\\
&\qquad\left.\left(\left(\sum_{t=k}^{m(k)}r(e_t)\right)\sum_{\substack{q\in\mathcal{M},\\ [\![q]\!](a_{1..m(k)})=e_{1..m(k)}}}\rho(q)\right)\right)\\
&\quad + s(m(k)-k+1)\sum_{\substack{q\in\mathcal{M},\\ [\![q]\!](a_{1..k-1})=e_{1..k-1}}}\rho(q)\\
&= pQ(\text{æ}_{1..k-1}, a_k) + s(m(k)-k+1)\sum_{\substack{q\in\mathcal{M},\\ [\![q]\!](a_{1..k-1})=e_{1..k-1}}}\rho(q)
\end{aligned}
$$

Therefore, Eq. (8) holds for

$$
c = (m(k)-k+1)\sum_{\substack{q\in\mathcal{M},\\ [\![q]\!](a_{1..k-1})=e_{1..k-1}}}\rho(q)
$$

■

Proof [Proof of Theorem 7]
From Lemma 22,

$$
\sum_{\substack{q\in\mathcal{M},\\ [\![q]\!](a_{1..m(k)})=e_{1..m(k)}}}\rho(q) \tag{16}
$$

is computable if $\mathcal{M}$ is a model of terminating computation.

From Lemmas 12, 13, and 17, $Q(\text{æ}_{1..k-1}, a)$ defined by Eq. (5) is computable, because the right-hand side of Eq. (5) only consists of addition, multiplication, maximization, and the right-hand side of (16). ■

Proof [Proof of Theorem 10] From Corollary 9 we can obtain an equivalent UAI algorithm satisfying $\mathcal{R} \subset (0, \frac{1}{m(k)-k+1})$ if we know the lower bound and the upper bound of the set of rewards.[3] From Theorem 7, the action value function in the $\arg\max$ operation in Eq. (6) is digit-wise computable. Since $\arg\max$ for any monomorphic function is computable from Lemma 20, it is enough to show that

$$
a \mapsto Q(\text{æ}_{1..k-1}, a)
$$

is monomorphic for any $\text{æ}_{1..k-1}$ with probability 1.

---

3. Note that $\mathcal{R}$ must not include 0 unlike Theorem 7.

From Eq. (5), we obtain

$$
\begin{aligned}
Q(\text{æ}_{1..k-1}, a_k) & \\
= \sum_{e_k \in \mathcal{E}} \max_{a_{k+1} \in \mathcal{A}} & \sum_{e_{k+1} \in \mathcal{E}} \max_{a_{k+2} \in \mathcal{A}} ... \max_{a_{m(k)} \in \mathcal{A}} \sum_{e_{m(k)} \in \mathcal{E}} \\
& \left(\left(\sum_{t=k}^{m(k)} r(e_t) \right) \sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..m(k)}) = e_{1..m(k)}}} \xi_2(q) \right) \\
= \max_{a_{k+1} \in \mathcal{A}} \max_{a_{k+2} \in \mathcal{A}} ... \max_{a_{m(k)} \in \mathcal{A}} & \left(\sum_{\substack{q \in \mathcal{M}, \\ [\![q]\!](a_{1..k-1}) = e_{1..k-1}}} \left(\sum_{t=k}^{m(k)} r(e_t) \right) \xi_2(q) \right) \qquad (17)
\end{aligned}
$$

where e_t in Eq. (17) for $t \geq k$ is obtained from $e_{1..m(k)} = [\![q]\!](a_{1..m(k)})$. Note that $\sum_{t=k}^{m(k)} r(e_t)$ is positive and rational because $m(k) \geq k$ and $\mathcal{R} \subset (0, \frac{1}{m(k)-k+1}) \cap \mathbb{Q}$.

Now, since from the premise $\mathcal{M}$ is equipped with a conditional construct, we can consider the following environment program candidate $q(b)$ for each $b \in \mathcal{A}$:

$$
[\![q(b)]\!](a_{1..i})
$$
$$
= \begin{cases} e_{1..i}, & \text{if } i < k; \\ e_{1..k-1} + replicate(m(k) - k + 1, (o, r_{\max})), & \text{if } i \geq k \text{ and } a_k = b; \\ e_{1..k-1} + replicate(m(k) - k + 1, (o, r_{\min})), & \text{if } i \geq k \text{ and } a_k \neq b; \end{cases}
$$

where

$$
replicate(n, x) = \underbrace{x : ... : x : [\,]}_{n}
$$

and $o \in \mathcal{O}$ can be chosen arbitrarily. $q(b)$ returns a fixed sequence $e_{1..k-1}$ without seeing the action sequence until time $k - 1$, but the time k is the judgment day. $a_k = b$ promises the eternal heaven, while other selections result in the eternal hell. Let us consider the behavior of the term on $q(b)$ in Eq. (17).

$q(b)$ satisfies $[\![q(b)]\!](a_{1..k-1}) = e_{1..k-1}$. The term on $q(b)$

$$
\left(\sum_{t=k}^{m(k)} r(e_t) \right) \xi_2(q(b)) = \begin{cases} (m(k) - k + 1) r_{\max} \xi_2(q(b)), & \text{if } a = b; \\ (m(k) - k + 1) r_{\min} \xi_2(q(b)), & \text{if } a \neq b. \end{cases}
$$

remains in the final $Q(\text{æ}_{1..k-1}, a)$, no matter which actions are selected as $a_{k+1} ... a_m(k)$ by the max operations. Since from the premise $r_{\min} \neq r_{\max}$, the coefficient for $\xi_2(q(b))$ takes different positive rational values depending on whether $a = b$ or not.

From the construction of ξ_2, the fraction $d(\eta(q(b)))$ of $\xi_2(q(b))$ is a random real value independent of the fraction $d(\eta(q_1))$ of $\xi_2(q_1)$ for any q_1 such that $q_1 \neq q(b)$. Thus, $Q(\text{æ}_{1..k-1}, a) \neq Q(\text{æ}_{1..k-1}, b)$ with probability 1 for any $a \neq b$.

Moreover, although the domain $(\mathcal{A} \times \mathcal{E})* \times \mathcal{A}$ of Q is infinitely countable when $(\mathcal{A} \times \mathcal{E})*$ is limited to finite lists, $a \mapsto Q(\text{æ}_{1..k-1}, a)$ is monomorphic for any $\text{æ}_{1..k-1}$ with probability 1,

because the Q value can be taken randomly from a continuum. ∎

References

Bird, R., and Wadler, P. 1988. *An Introduction to Functional Programming*. Prentice-Hall.

Boehm, H., and Cartwright, R. 1990. Exact Real Arithmetic Formulating Real Numbers As Functions. In Turner, D. A., ed., *Research Topics in Functional Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. 43–64.

Claessen, K., and Pałka, M. H. 2013. Splittable Pseudorandom Number Generators Using Cryptographic Hashing. In *Proceedings of the 2013 ACM SIGPLAN Symposium on Haskell*, Haskell '13, 47–58. New York, NY, USA: ACM.

Hutter, M. 2007. Universal Algorithmic Intelligence: A Mathematical Top→Down Approach. In Goertzel, B., and Pennachin, C., eds., *Artificial General Intelligence*, Cognitive Technologies. Berlin: Springer. 227–290.

Katayama, S. 2016. Ideas for a Reinforcement Learning Algorithm that Learns Programs. In *Artificial General Intelligence - 9th International Conference, AGI 2016, AGI 2016, New York, USA, July 16–19, 2016, Proceedings*, 354–362.

Plume, D. 1998. *A Calculator for Exact Real Number Computation*. Ph.D. Dissertation, University of Edinburgh.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Veness, J.; Ng, K. S.; Hutter, M.; Uther, W.; and Silver, D. 2011. A Monte-Carlo AIXI Approximation. *Journal of Artificial Intelligence Research* 40:95–142.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. In *Machine Learning*, 279–292.

Willems, F. M. J.; Shtarkov, Y. M.; and Tjalkens, T. J. 1995. The Context Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory* 41:653–664.